WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Friday, September 17, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L34	l29 or l31 or l32	36
☐	L33	L26 and l21	0
☐	L32	L26 and l20	3
☐	L31	L26 and l15	1
☐	L30	L26 and l13	0
☐	L29	L26 and l10	35
☐	L28	L26 and l9	0
☐	L27	L26 and l2	0
☐	L26	20001219	485
☐	L25	711/146.ccls.	636
☐	L24	20001219	11
☐	L23	l20 and (snoop or snooping)	15
☐	L22	20001219	5
☐	L21	(request adj3 controller) near8 (buffer or buffering or queue) near8 (remove or removing)	8
☐	L20	(request or (request adj3 controller)) near8 (buffer or buffering or queue) near8 (remove or removing) near8 response	62
☐	L19	L15 and (snoop or snooping)	2
☐	L18	L17 and (snoop or snooping)	0
☐	L17	20001219	25
☐	L16	L15 and l10	27
☐	L15	(status adj2 signal) near8 interval	305
☐	L14	L13 and l10	1
☐	L13	(status adj2 signal) near8 latency	9
☐	L12	(status adj2 signal) near8 (cannot adj3 received) near8 latency	0
☐	L11	L10 and l9	1
☐	L10	(request or queue) near8 (remove or removing)	6924
☐	L9	(status adj2 signal) near8 (cannot adj3 received)	10
☐	L8	20001219	6
☐	L7	(snoop or snooping) near8 (without adj3 response)	7
		(request adj2 queue) near8 (remove or removing) near8 (without adj3	

☐	L6	response)	0
☐	L5	20001219	4
☐	L4	l2 and (snoop or snooping)	8
☐	L3	20001219	12
☐	L2	(request adj2 queue) near8 (remove or removing) near8 (response)	25
☐	L1	request-and-forget	1

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Search Forms
Search Results
Help
User Searches
Preferences
Logout

Generate Collection

L34: Entry 26 of 36 File: USPT Oct 13, 1998

DOCUMENT-IDENTIFIER: US 5822765 A
TITLE: System and method for resolving contention arising from execution of cache coherency operations in a multiple cache computer system

Application Filing Date (1):
19951214

Current US Original Classification (1):
711/146

CLAIMS:

3. The system of claim 2, wherein, in a responding controller which has loaded the buffer in response to a certain operation, the responding controller logic circuitry toggles a status bit of the associated buffer to a "busy" status if a retry signal is not received during the predetermined time interval.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L8: Entry 6 of 6 File: USPT May 12, 1998

DOCUMENT-IDENTIFIER: US 5752265 A
TITLE: Memory accessing in a multi-processor system using snooping

Application Filing Date (1):
19960613

Detailed Description Text (20):
Each of the level two cache controllers 23, 25 responds to the snoop request as soon as possible. Each response is forwarded by the respective distributed controller to the central controller 70. The central host bus controller 70 monitors the snoop responses individually in the order they are received to determine the result of the snoop request. In this manner, the central controller 70 determines and controls when to terminate the snoop routine, as further explained below. Once the result is known, it is conveyed to the distributed controller 26. Depending upon the received snoop responses, the snoop routine may be interrupted such that the CPU 10 is allowed to continue its cycle to memory without awaiting snoop responses from CPUs which have not yet responded.

Detailed Description Text (21):
The central controller 70 initially determines whether a received snoop response indicates that a cache memory is in either a shared or exclusive state with respect to the memory address to be accessed, as indicated by 305. If the determination is affirmative, the result is conveyed to the distributed controller 26 which instructs the CPU 10 to continue its cycle to memory without awaiting further snoop responses from other CPUs, as indicated by step 313. In other words, the central controller 70 interrupts the snoop routine and permits the CPU 10 to continue its cycle to memory.

Detailed Description Text (22):
If the central controller 70 determines that the received snoop response does not indicate that a cache is in either a shared or exclusive state with respect to the memory address for which access is requested, then the central controller 70 determines whether the received snoop response indicates that a cache is in a modified state with respect to the memory address to be accessed, as shown by 307. If the central controller 70 determines that the received snoop response indicates that a cache is in a modified state with respect to the memory address to be accessed by the CPU 10, then the result is conveyed to the distributed controller 26. The central controller 70 gives the CPU 11 access to the bus 40, and the CPU 11 performs a write back operation with respect to the modified data stored in the specified memory address, as indicated by step 311. After the CPU 11 completes its write back operation, the distributed controller 26 reacquires the bus 40 so that the CPU 10 can continue and complete its read or write cycle to memory without awaiting further snoop responses, as shown by the step 313.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L8: Entry 5 of 6 File: USPT Nov 23, 1999

DOCUMENT-IDENTIFIER: US 5991855 A
** See image for Certificate of Correction **
TITLE: Low latency memory read with concurrent pipe lined snoops

Application Filing Date (1):
19970702

Detailed Description Text (49):
Based on the foregoing discussion it will be appreciated that the embodiments of the present invention described herein provide a highly efficient method of executing memory transactions in a computer system. The method transmits a memory request of a transaction to the system memory without waiting for a response to a corresponding snoop request of the transaction. As used herein, the term "snoop request" is intended broadly to include a request from any computer device to another computer device to determine whether a memory device stores information for any address corresponding to a selected address of another memory device being requested by a corresponding memory request. This allows memory transactions to be completed more quickly than do prior art methods that require the snoop phase of a transaction to be completed before a memory request can be sent to the system memory. Moreover, the method enables snoop requests to be processed in a pipelined manner to execute the snoop phases of transactions more quickly which also reduces the ability of snoop requests to delay the execution of transaction requests issued by the system processor.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L8: Entry 3 of 6 File: USPT Oct 31, 2000

DOCUMENT-IDENTIFIER: US 6141735 A
TITLE: Performing a memory access cycle in a multi-processor computer system

Application Filing Date (1):
19980429

Detailed Description Text (21):
Each of the level two cache controllers 23, 25 responds to the snoop request as soon as possible. Each response is forwarded by the respective distributed controller to the central controller 70. The central host bus controller 70 monitors the snoop responses individually in the order they are received to determine the result of the snoop request. In this manner, the central controller 70 determines and controls when to terminate the snoop routine, as further explained below. Once the result is known, it is conveyed to the distributed controller 26. Depending upon the received snoop responses, the snoop routine may be interrupted such that the CPU 10 is allowed to continue its cycle to memory without awaiting snoop responses from CPUs which have not yet responded.

Detailed Description Text (22):
The central controller 70 initially determines whether a received snoop response indicates that a cache memory is in either a shared or exclusive state with respect to the memory address to be accessed, as indicated by 305. If the determination is affirmative, the result is conveyed to the distributed controller 26 which instructs the CPU 10 to continue its cycle to memory without awaiting further snoop responses from other CPUs, as indicated by step 313. In other words, the central controller 70 interrupts the snoop routine and permits the CPU 10 to continue its cycle to memory.

Detailed Description Text (23):
If the central controller 70 determines that the received snoop response does not indicate that a cache is in either a shared or exclusive state with respect to the memory address for which access is requested, then the central controller 70 determines whether the received snoop response indicates that a cache is in a modified state with respect to the memory address to be accessed, as shown by 307. If the central controller 70 determines that the received snoop response indicates that a cache is in a modified state with respect to the memory address to be accessed by the CPU 10, then the result is conveyed to the distributed controller 26. The central controller 70 gives the CPU 11 access to the bus 40, and the CPU 11 performs a write back operation with respect to the modified data stored in the specified memory address, as indicated by step 311. After the CPU 11 completes its write back operation, the distributed controller 26 reacquires the bus 40 so that the CPU 10 can continue and complete its read or write cycle to memory without awaiting further snoop responses, as shown by the step 313.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L19: Entry 2 of 2 File: USPT Oct 13, 1998

DOCUMENT-IDENTIFIER: US 5822765 A
TITLE: System and method for resolving contention arising from execution of cache coherency operations in a multiple cache computer system

Brief Summary Text (5):
In such a distributed, hierarchical system, information required to resolve execution of an operation may not be available to a responding device within a time interval desired for a response by the device. For example, in a system with cache controllers for managing cache memory and handling processor transactions, a processor may communicate to the system bus through a cache controller. The cache controller may assist the processor by reducing processor traffic on the system bus by the cache controller monitoring ("snooping") the system bus and filtering out transactions ("snoop filtering") which the processor does not need to process. The cache controller may also automatically maintain coherence of information residing in cache memory through hardware that supports the snooping transactions on the system bus and processes coherency status information.

Brief Summary Text (12):
The embodiment further contemplates that when a first cache controller on the bus sends a certain request and the buffer of a second cache controller on the bus is marked "not busy", the second cache controller snoops the request into the buffer. If a third cache controller on the bus outputs a retry signal, the second cache controller halts further processing of the certain request so that when the certain request is repeated by the first cache controller in response to the retry signal from the third cache controller, the certain request will be loaded into the second cache controller buffer without the second cache controller generating a retry signal, and so that no cache controller receiving the certain request fully processes the request until all the cache controllers receiving the request can fully process the request.

Detailed Description Text (2):
Referring to FIG. 1, a system 10 embodying the present invention has a processor 12.1 with a first level (L1) cache 14.1 for storing blocks of memory 16 from a main memory 18. The L1 cache 14.1 has an instruction cache portion 14a.1 and a separate data cache portion 14b.1. The processor 12.1 also is associated with and uses a second level (L2) memory cache 20.1 that is larger than the L1 cache 14.1. The blocks 22a.1 and 22b.1 (also referred to as "lines") of memory contained in the L1 cache 14.1 are maintained as a subset of the blocks 24.1 in the L2 cache 20.1. The processor 12.1 is connected by a processor bus 27.1 to an L2 cache controller 26.1. The L2 cache controller 26.1 is connected to the L2 cache 20.1 by an L2 cache bus 29.1 and to the processor 12.1 by the processor bus 27.1. The L2 cache controller 26.1 manages and services the L2 cache 20.1 and handles processor 12.1 transactions. The controller 26.1 is interposed between the processor 12.1 and a system bus 28, and is therefore referred to as a "in-line" cache controller. The processor 12.1 is thus coupled to the system bus 28 and other devices coupled thereto, including the L2 memory cache 20.1, through the in-line L2 cache controller 26.1 and the buses 27 and 29. A second L2 cache controller 26.2 is also connected to the system bus 28, and to associated processor 12.2, L1 cache 14.2, and L2 cache 20.2. Likewise, a third L2 cache controller 26.3 is connected to the

system bus 28, and associated processor 12.3, L1 cache 14.3, and L2 cache 20.3. There may be up to n L2 cache controllers 26.n, processors, 12.n, etc. These devices may be referred to herein collectively or singly, for example, as "L2 cache controllers 26", "processors 12", or "an L2 cache controller 26", "a processor 12", etc. For this multi-processor application, an in-line cache L2 controller 26, such as controller 26.1, assists its associated processor 12, such as processor 12.1 associated with L2 cache controller 26.1, by reducing traffic on the processor bus 27. This is done by a cache controller 26 monitoring ("snooping") the system bus 28 and filtering out transactions ("snoop filtering") which the associated processor 12 does not need to process. For example, associated processor 12 does not need to process transactions to addresses not stored within the L1 cache 14. The cache controller 26 also maintains memory coherence through snooping transactions on the system bus 28 and processing coherency status information based on the nature of the snooped transactions and the state of affected blocks 22b in the associate data cache 14 and blocks 24 in the associate L2 cache 20. An L2 cache controller 26 has a first snoop buffer 23 and a second snoop buffer 25 so that certain snooped transactions may be stored by the cache controller 26 for processing.

Detailed Description Text (6):
This protocol may be implemented according to the following signals. An ICBI operation is an "address only" transaction which uses address ("ADDR"), transaction type ("TT"), transfer start ("TS"), address acknowledge ("AACK") and address retry ("ARTRY") lines on the system bus 28. The controller 26.1 initiates a first ICBI operation by outputing TT, ADDR and TS signals. The TS signal notifies other devices on the bus, e.g., controllers 26.2 and 26.3, that the TT and ADDR signals are valid and may be snooped during the current and next clock cycles. Since this is the first ICBI operation the snoop buffer 25.2 of the second cache controller 26.2 is not busy, and the controller 26.2 loads the ICBI address into the buffer 25.2 without accessing the directory to check for a hit. Likewise, the snoop buffer 25.3 of the third cache controller 26.3 is not busy, and the third controller 26.3 loads the ICBI address into its snoop buffer 25.3.

Detailed Description Text (7):
Once a snoop buffer in a L2 cache controller contains an ICBI address for sending to the processor, the controller will send and resend the address to the processor associated with the controller until the controller receives a signal from the processor indicating the instruction has been accepted by the processor. From the time that the ICBI address has been loaded into the snoop buffer until the time that the controller is notified that the processor has accepted the address, the buffer is marked busy.

Detailed Description Text (8):
When the first controller 26.1 places another ICBI operation on the system bus 28 at time Tb, suppose that the second controller 25.2 has not been notified by the processor 12.2 that the first ICBI operation has been accepted so that snoop buffer 25.2 for the second controller 26.2 is still busy. In response to the TT, ADDR and TS signals asserted at time Tb, an AACK signal is generated at time Tc by a system arbiter (not shown) indicating to the sending cache controller 26.1 that the ADDR signals have been detected and notifying the other devices coupled to the bus 28, such as cache controllers 26.2 and 26.3, that an ARTRY signal must be asserted before a certain succeeding cycle at time Te, if at all. Since the snoop buffer 25.2 of cache controller 26.2 is busy at the time Tb of the second ICBI operation, the controller 26.2 asserts an ARTRY signal at time Td, before the time Te. Then, if the controller 26.1 resends the second ICBI operation again at some time after Tf the operation can accepted and can be cleanly executed.

Detailed Description Text (9):
Certain features and advantages of the preferred embodiment of the present invention as shown in FIGS 1, 2, 5 and 6 may be further understood by comparison to the system shown in FIG. 3. (FIG. 3 uses the same identifying numbers as used in

FIG. 1 for elements that are in common with the system of FIG. 1.) An L1 cache inclusion bit 30 is maintained in a directory 32 in an L2 cache controller 26 for identifying whether a block 24 of memory in the L2 cache 20 has been fetched into the L1 cache 14. An L1 instruction cache inclusion bit 34 in the L2 cache directory 32 designates whether a block 24 of memory was sent to the L1 cache 14 as a result of an instruction fetch. If the block 24 is sent to data cache 14b then its coherence will be automatically maintained by other aspects of the system and there is no need for the cache controller 26 to respond to ICBI instructions from other processors 12 or controllers 26. However, if the block 24 is sent to an instruction cache 14a then the hardware will not automatically maintain memory coherence, and the controller 26 must respond to snooped ICBI instructions. That is, when cache controller 26 snoops an ICBI instruction on the system bus 28, the controller 26 determines whether the block 24 identified by the address in the ICBI instruction corresponds to a block 22a or 22b of memory in the L1 cache 14 by referring to the L1 cache inclusion bit 30 in the L2 cache directory 32. (This mechanism wherein a first processor or cache controller monitors the system bus and detects an operation from a second processor or controller which affects the first processor or controller is referred to as a "snoop hit"). If there is a snoop hit then the cache controller 26 checks an L1 instruction cache inclusion bit 34 in the L2 directory 32 to determine whether the block 24 has previously been sent to the processor L1 instruction cache 14a so that a corresponding block 22 may reside in the cache 14a. If so, then the controller 26 sends the ICBI instruction to the processor 12, and the controller 26 clears its instruction cache inclusion bit 34. If the processor 12 determines that the block 22 is still in the L1 instruction cache 14a then the processor 12 updates an L1 cache directory (not shown) to indicate that the block 22 is invalid.

Detailed Description Text (11):
Another system does not require certain status information as described above to be kept in the directory 32 for the purpose of instruction coherence, but is limited because it may result in devices in the system 10 encountering a deadlock condition. This non-inclusion bit based, but deadlock-prone system uses the single snoop buffer 23 in the L2 cache controller 26 in the system of FIG. 1, but does not use the instruction cache inclusion bit 34 shown in FIG. 3.

Detailed Description Text (12):
FIG. 4 illustrates the protocol and timing of signals for the non-inclusion bit based version of the system shown in FIG. 3. For this system a deadlock may occur for example, when the second controller 26.2 snoop buffer 23.2 is busy and the third controller 26.3 snoop buffer 23.3 is free and the controller 26.1 for the first processor 12.1 outputs an ICBI operation to the bus 28 at time Ta. At time Tb the system arbiter (not shown) generates an AACK signal in response to the ICBI request. Also in response to the ICBI request asserted at time Ta, the third controller 26.3 loads the ICBI instruction into its snoop buffer 23.3 at time Tc. Also, the second controller 26.2 detects the ICBI operation from the first controller 26.1 and detects the busy snoop buffer 23.2, so that the second controller 26.2 asserts an address retry signal ("ARTRY") on the bus 28 to the first controller 26.1 at time Td (prior to a certain predetermined time following the AACK signal).

Detailed Description Text (13):
Meanwhile, assume the buffer of 23.2 of the second controller 26.2 has become free at time Te. In response to the ARTRY signal, the first controller 26.1 sends the ICBI operation again at time Tf. At time Tg the system arbiter (not shown) generates an AACK signal in response to the ICBI operation. Also in response to the ICBI instruction repeated at time Tf, the second controller 26.2 loads the instruction into buffer 23.2 at time Th. Meanwhile, the third controller 26.3 asserts an ARTRY signal at time Ti because the third controller 26.3 snoop buffer 23.3 is still busy due to the ICBI operation issued at time Ta by the first controller 26.1. Assume that at time Tj the buffer 23.3 of the third controller

26.3 has become free. The first controller repeats the ICBI operation at time Tk in response to the ARTRY signal asserted at time Ti by the third controller 26.3. In response to the ICBI operation, the system arbiter (not shown) generates an AACK signal at time Tl and the controller 26.3 loads the ICBI address into the buffer 23.3 at time Tm, but The second controller 26.2 snoop buffer 23.2 is busy with the ICBI instruction issued at time Tf so that at time Tn the second controller 26.2 issues another ARTRY signal.

Detailed Description Text (15):
The preferred embodiment of the present invention avoids the memory requirements of the inclusion bit based version of the system shown in FIG. 3 and has certain features for avoiding the deadlock which may occur with the non-inclusion bit based version of the system shown in FIG. 3. Referring now to FIG. 5, an L2 controller of the present invention is illustrated in more detail than in FIG. 1. In the system 10 of FIGS. 1 and 5 the L2 cache controllers 26 have a first snoop buffer 23 (not shown in FIG. 5) and second snoop buffer 25 (the "ICBI snoop buffer"), wherein the second buffer 25 is for storing ICBI operations. The ICBI snoop buffer 25 reduces the likelihood of occurrence of a deadlock such as described for the non-inclusion bit based system of FIG. 3, since the ICBI snoop buffer 25 will not be busy with non-ICBI operations. To eliminate the possibility of a deadlock, L2 cache controllers 26 according to the present invention have certain additional elements and implement a certain protocol.

Detailed Description Text (16):
With regard to receiving and processing of an ICBI request, a controller 26 has a decoder 40 receiving TT signal inputs from the system bus 28 TT lines 42 and outputting a ICBI decode signal over a decode indication line 44 to an ICBI buffer control logic element 46 in response to an ICBI request on the bus 28. The logic element 46 inputs and outputs ARTRY signals to and from system bus 28 retry lines 48 and has a busy status bit 50 for indicating whether the ICBI buffer is busy. The buffer control logic element 46 outputs a load address control signal to the ICBI buffer 25 over a control signal line 52 and an ICBI request signal to a processor bus arbitration and snoop logic element 54 ("processor bus arbiter") over request/request complete signal lines 56. The buffer control logic 46 also inputs an ICBI request complete signal from the arbiter 54 over the line 56. The ICBI buffer 25 inputs and stores address signals from address lines 58 to the system bus and outputs a stored address to a multiplexing unit (mux) 60 over buffer address lines 62. The mux 60 inputs an address select control signal from the arbiter 54 over a select control line 64. In response to this address select control signal, the mux 60 reads the ICBI address from the ICBI buffer 25 and outputs the address over address lines 66 to the processor bus 27. The processor bus arbitration and snoop logic element 54 outputs bus control signals over control lines 68 and inputs ARTRY signals over retry lines 70 connecting the element 54 to the processor bus 27 in order to control communication between the L2 cache controller 26 and the processor 12.

Detailed Description Text (21):
The timing diagram of FIG. 6 illustrates certain signals of a complete ICBI instruction cycle for the system of FIGS. 1 and 5. The time line shown in FIG. 6 is not to scale and the time periods between times shown, such as Ta and Tb for example, are not necessarily the same as the time period between other times, such as Tb and Tc for example. For simplicity some signals described in the discussion of FIG. 5, such as the load address control signal which is output by control element 46, are not shown in FIG. 6. FIG. 6 particularly shows, in contrast to FIG. 5, how a second and third L2 cache controller of the present invention successfully execute an ICBI instruction requested by a first L2 cache controller without encountering deadlock. Beginning again with the circumstance as described in connection with FIG. 4, the second controller 26.2 snoop buffer 25.2 is busy and the third controller 26.3 snoop buffer 25.3 is free and the controller 26.1 for the first processor 12.1 outputs an ICBI operation to the bus 28 at time Ta. Outputting

an ICBI operation includes the controller 26.1 asserting TS, TT and Addr signals on the bus 28, wherein the Addr signals designate a block of information affected by the ICBI operation, the TS signal indicates the start of a transfer, and the TT signals indicate that the type of transfer is an ICBI operation. At time Tb the system arbiter (not shown) generates an AACK signal in response to the ICBI request. Also in response to the ICBI request asserted at time Ta, the third controller 26.3 loads the ICBI address into its snoop buffer 25.3 at time Tc. The controller 26.3 element 46.3, however, waits for a certain time for a possible ARTRY signal on the bus 28 before requesting the processor bus arbiter 54.3 to process the ICBI operation (i.e., before posting the request for further processing by the arbiter 54.3 and, ultimately, the processor 12.3). Meanwhile, the second controller 26.2, in response to the ICBI request asserted at time Ta, detects the busy snoop buffer 25.2, and the second controller 26.2 asserts an address retry signal ("ARTRY") on the bus 28 at time Td, which is during the clock cycle following the AACK signal, or more generally stated, is prior to a certain predetermined time following the AACK signal and during the time that the controller 26.3 element 46.3 waits for a possible ARTRY signal. Then, assume the buffer of 25.2 of the second controller 26.2 becomes free at time Te. The element 46.3 cancels the posting of the request in response to the ARTRY signal asserted at time Td, and thereby does not request the arbiter 54.3 to process the ICBI address in the buffer 25.3 at time Tf. Also, in response to the ARTRY signal, the first controller 26.1 sends the ICBI operation again at time Tg. At time Th the system arbiter (not shown) generates an AACK signal in response to the ICBI operation. Then, in response to the ICBI instruction repeated at time Tg, the second controller 26.2 and the third controller 26.3 both load the ICBI address into buffers 25.2 and 25.3 respectively at times Ti and Tj. The controllers 26.2 and 26.3 then wait for a certain time for a possible ARTRY signal on the bus 28 before requesting the arbiters 54.2 and 54.3 to process the ICBI operation. This time, however, no ARTRY signal is asserted because neither of the buffers 25.2 and 25.3 were busy when the ICBI operation was requested the second time at time Tg. After the certain time has elapsed and no ARTRY signal has occurred the controllers 26.2 and 26.3 elements 46.2 and 46.3 set the status bits 50.2 and 50.3 to indicate the buffers 25.2 and 25.3 are busy, and post the ICBI request to the arbiters 54.2 and 54.3 by outputting an ICBI request signal over lines 56.2 and 56.3 at times Tk and Tl respectively. Deadlock has thus been avoided due to the structure and protocol of the system 10 of FIGS. 1, 5 and 6.

CLAIMS:

3. The system of claim 2, wherein, in a responding controller which has loaded the buffer in response to a certain operation, the responding controller logic circuitry toggles a status bit of the associated buffer to a "busy" status if a retry signal is not received during the predetermined time interval.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 2 of 5 File: USPT Aug 7, 2001

DOCUMENT-IDENTIFIER: US 6272600 B1
TITLE: Memory request reordering in a data processing system

Application Filing Date (1):
19970228

Detailed Description Text (44):
The shift controller 751 is coupled to know the contents of the request buffer 703. After a request has been scheduled, the shift controller 751 removes that request from the request buffer 703. To do this, the shift controller 751 performs two functions. First, the shift controller 751 deduces which request was the one that was just scheduled by identifying the oldest target-available request that matches the type of request just scheduled (i.e., read type 747 or write type 749). Second, the shift controller 751 instructs the request buffer 703 to shift all requests older than the just-scheduled request by one element, away from the input end 705 (i.e., newer end) of the request buffer 703.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L22: Entry 5 of 5 File: USPT May 11, 1993

DOCUMENT-IDENTIFIER: US 5210860 A
TITLE: Intelligent disk array controller

Application Filing Date (1):
19900720

Detailed Description Text (41):
If in step 512, it is determined that the next action to be taken by the task 500 is to check the results of the read request, control transfers to step 534 (FIG. 5C). Control transfer to step 536, wherein the processor 122 reads the transfer buffer to determine if an error code indicating disk sector failure has been returned by the read request. If a read request has an error code returned, control transfers to step 538. An error code will be returned if any one of the sectors within the track specified in the read request is faulty. In step 538 the task 500 notifies the controller of the disk failure. The failure notification may be acted upon locally by the disk controller 112 or may be passed on to the host system via BMIC 118 to permit the operator or operating system to take any corrective action necessary. One means of corrective action which may be taken locally is the automatic remapping of a faulty sector to a new sector on the same disk. This method of automatically remapping a faulty disk sector is described in co-pending U.S. application Ser. No. 556,646 to Ewert et al., for AUTOMATIC HARD DISK BAD SECTOR REMAPPING, filed Jul. 20, 1990 and assigned to Compaq Computer Corporation. However, it is understood that other methods of corrective action, such as manual intervention and regeneration of the data contained within the faulty disk sector may be taken. Control of the processor 122 transfers to step 540. If its is determined in step 536 that the completed read request did not return an error code, control transfers to step 540 in which the processor 122 removes this current read request from the disk controller 112 queue. This is necessary to permit the memory 128 which has been allocated to the read request to be deallocated. If the read request is not dequeued, the local memory 128 could not be deallocated and would rapidly be filled with completed read requests, as well a new requests and data to be transferred. Control transfers to step 542 wherein the processor 122 deallocates the memory 128 which was allocated for return information from the drive request. Control transfers to step 544 wherein processor 122 sets the NEXT.sub.-- ACTION variable to issue a new read request.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L24: Entry 1 of 11 File: USPT Nov 26, 2002

DOCUMENT-IDENTIFIER: US 6487643 B1
TITLE: Method and apparatus for preventing starvation in a multi-node architecture

Application Filing Date (1):
20000929

Brief Summary Text (5):
In many computer systems, the set of data currently being used by a microprocessor may be copied from a system memory device such as a dynamic random access memory (DRAM) into a relatively smaller but faster cache memory device such as a static random access memory (SRAM). The cache memory device is usually private to each processor such that only one processor can read or write to it. In such systems, a cache is said to be "coherent" if the information resident in the cache reflects a consistent view of the information in all the private cache memory devices and the DRAM memory. Cache "snooping" is a technique used to detect the state of a memory location in private cache memory devices on a memory access that might cause a cache coherency problem. In a multi-processor system, the messages sent between processors may include cache snooping messages.

Detailed Description Text (4):
The nodes in system 100 may send messages that are directed to a processor, memory or resource in another node. For example, one node may send a request to read from a memory location that is stored in another node. Similarly, a node may send a request to snoop the caches in the other nodes. In one embodiment, all requests in system 100 from one node to another node may be sent to switching agent 140, and switching agent 140 may send requests to other nodes that are based on the first request. For example, switching agent 140 may receive a request from a first node to snoop for a particular memory location, and switching agent 140 may send snoop requests to the other nodes in system 100 as is appropriate to carry out the received snoop request.

Detailed Description Text (6):
A request is associated with a memory location, for example, if it is a request to access that location in the memory. Thus, a request to read from a location in a memory or to write to a location in a memory are associated with that location in the memory. Other types of requests also may be associated with a memory location, such as for example a snoop request.

Detailed Description Text (15):
In a further embodiment, inter-node communication in system 100 is asynchronous (i.e., there is no fixed timing between events). In a still further embodiment, inter-node communication is sent in the form of packets which may contain a header and data sections. An example of a message size may be 144 bits. In an embodiment, the messages sent may include requests and responses. In a further embodiment, the types of requests that the nodes may send and receive may include a memory read request, memory write request, cache snoop request, cache flush request, memory update request, cache line replacement request, input/output port read request, and input/output port write request. Requests may contain fields such as a packet type, destination ID, request type, source ID, transaction address, request length,

stream ID, and ordering semantics.

Detailed Description Text (16):
In an embodiment of the present invention, the processors in nodes 110, 120 and 130 may be shared memory multi-processors, and each of the memories 119, 129, and 139 may be part of the same shared physical address space. In a further embodiment, the processors in nodes 110, 120, and 130 communicate with each other through shared memory reads and writes (i.e., by writing to and reading from memory 119, 129 and 139). In a further embodiment, the processors in nodes 110, 120 and 130 each have one or more caches (e.g., Level 1 and Level 2 caches), and these caches are kept coherent using the switching agent 140. For example, when processor 111 accesses a location in memory 119, it may send a snoop request for that memory location to switching agent 140, which may determine if any of the processors in second node 120 and third node 130 have cached that memory location. A snoop request may be generated when a processor needs other processors in the system to look in their own caches to see if a particular line is present in their cache.

Detailed Description Text (19):
Switching agent 140 may process requests as follows. If switching agent 140 receives from third node 130 a request to snoop the memory location designated as A (in FIG. 2), memory manager 149 may determine from table 143 that memory location A is cached in both cache 113 (in first node 110) and cache 117 (in second node 120). Memory manager 149 may then cause snoop requests that are associated with location A to be sent to first node 110 and second node 120.

Detailed Description Text (21):
FIG. 4 is a flow diagram of a method of managing requests in a multi-node system according to an embodiment of the present invention. This method may be performed by a device, such as for example switching agent 140 of FIG. 1. As shown in FIG. 4, a new request is received by the device (401). The new request may be received from a first node, such as first node 110, and the new request may be associated with a location in a memory address space, such as memory address space 201. For example, first node 110 may have sent a request to read from a location in the memory address space or a request to snoop for a location in the memory address space. Prior to receiving the new request, switching agent 140 may have sent one or more requests to nodes in the system. During the process of sending requests for each memory location, switching agent 140 may have created a new entry associated with the memory location in a response pending buffer. The response pending buffer may contain entries that are associated with memory locations for which requests have been sent by the device and for which the device has not received all the responses (i.e., a response is pending). That is, an entry may be created in the response pending buffer when one or more requests are sent and may be removed from the response pending buffer when all responses to the requests are received.

Detailed Description Text (23):
If a Pending Request Buffer entry is available to process the new request, then the new request is stored in the Pending Request Buffer (403). Once a Pending Request Buffer entry is allocated to the new request, the switching device 140 may check the response pending buffer to identify entries associated with the same location in the memory address space as the new request (404-405). If an entry in the response pending buffer is identified as being associated with the same location as the new request, then the switching agent determines (1) if an identified entry is associated with a request that was sent to the same node as the node from which the new request was received and (2) if a response associated with the request has not been received (406). If the identified entry has an outstanding request to the node from which the new request was received, then the request is removed from the Pending Buffer and a retry response is sent to that node requesting resending of the new request (407-408). For example, if (1) the new request was received from the first node, (2) a second request associated with the same memory location as the new request had been sent by the switching agent 140 to the first node, and (3)

a response to the second request has not been received by the switching agent 140, then the switching agent 140 removes the request from the pending response buffer (407) and sends a retry response to the first node requesting that the first node resend the new request (408). This type of request retry is classified as a "conflict induced retry", where a conflicting entry to the same memory location in the Response Pending Buffer causes the request to be rejected from the switching agent. At a later time, the switching agent 140 may receive a resent version of the new request, and the received resent version may be processed according to the same method. If none of the entries in the response pending buffer are associated with the same location as the received resent request and if the Response Pending Buffer is not full, then the new request may be entered in the Response Pending Buffer and may be processed as discussed below.

Detailed Description Text (26):
If the pending request buffer is not full, then the new request may be further processed. Processing the request may include identifying nodes that contain a copy of the location associated with the request in their private cache memory, sending snoop requests that are based on the new request to each of the identified nodes (503), and updating the entry in the response pending buffer to indicate that a response is pending from each of the identified nodes (504). For example, if a new request to read from a location that is contained in memory 129 is received from first node 110, a memory manger in the switching device may check a table and determine that a request to read from that location should be sent to second node 120. A snoop request to that location may be sent to the second node 120, and this request may be based on the received new request. In addition, the entry associated with the request stored in the response pending buffer 142 may be updated to include a destination node (second node 120), an ID of the associated request, and the location in the memory address space that is associated with the request.

Detailed Description Text (27):
When the switching agent 140 receives a response associated with an entry in the response pending buffer 142, it updates the entry to indicate that the response has been received from the associated node. The switching agent may remove the entry from the response pending buffer after responses have been received for all the requests sent to different nodes for that entry (505-507). Once an entry is removed from the response pending buffer, the associated entry in the pending request buffer may also be removed and the entry is available for another new request (508). The switching agent 140 may then select an entry for processing from a pending requests buffer, such as pending requests buffer 145.

CLAIMS:

5. The method of claim 4, wherein the third request is a request to snoop for said location in the memory address space.

9. The method of claim 8, wherein each of said identified nodes includes a cache memory device, wherein said received new request is a request to read from said location in the memory address space, and wherein the requests sent to each of said identified nodes are requests to snoop said cache included in that node for said location in the memory address space.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L24: Entry 2 of 11 File: USPT Jul 9, 2002

DOCUMENT-IDENTIFIER: US 6418514 B1
TITLE: Removal of posted operations from cache operations queue

Application Filing Date (1):
19980217

Brief Summary Text (10):
To implement cache coherency in a system, the processors communicate over a common generalized interconnect (i.e., bus 20). The processors pass messages over the interconnect indicating their desire to read or write memory locations. When an operation is placed on the interconnect, all of the other processors"snoop" (monitor) this operation and decide if the state of their caches can allow the requested operation to proceed and if so, under what conditions. There are several bus transactions that require snooping and follow-up action to honor the bus transactions and maintain memory coherency. The snooping operation is triggered by the receipt of a qualified snoop request, generated by the assertion of certain bus signals. Instruction processing is interrupted only when a snoop hit occurs and the snoop state machine determines that an additional cache snoop is required to resolve the coherency of the offended sector.

Brief Summary Text (12):
For example, consider a processor, say 12a, attempting to read a location in memory. It first polls its own L1 cache (24 or 26). If the block is not present in the L1 cache, the request is forwarded to the L2 cache (30). If the block is not present in the L2 cache, the request is forwarded on to lower cache levels, like the L3 cache. If the block is not present in the lower level caches, the request is then presented on the generalized interconnect (20) to be serviced. Once an operation has been placed on the generalized interconnect, all other processing units snoop the operation and determine if the block is present in their caches. If a given processing unit has the block of data requested by processing unit in its L1 cache, and that data is modified, by the principle of inclusion the L2 cache and any lower level caches also have copies of the block (however, their copies are stale, since the copy in the processor's cache is modified). Therefore, when the lowest level cache (e.g., L3) of the processing unit snoops the read instruction, it will determine that the block requested is present and modified in a higher level cache. When this occurs, the L3 cache places a message on the generalized interconnect informing the processing unit that it must "retry" its operation again at a later time, because the actual value of the memory location is in the L1 cache at the top of the memory hierarchy and must be retrieved to make it available to service the read request of the initiating processing unit.

Brief Summary Text (14):
The initiating processing unit eventually re-presents the read request on the generalized interconnect. At this point, however, the modified data has been retrieved from the L1 cache of a processing unit and the read request from the initiating processor will be satisfied. The scenario just described is commonly referred to as a "snoop push". A read request is snooped on the generalized interconnect which causes the processing unit to "push" the block to the bottom of the hierarchy to satisfy the read request made by the initiating processing unit.

Brief Summary Text (17):
When a block is in the Modified state, the addressed block is valid only in the cache having the modified block, and the modified data has not been written back to system memory. When a block is Exclusive, it is present only in the noted block, and is consistent with system memory. If a block is Shared, it is valid in that cache and in at least one other cache, all of the shared blocks being consistent with system memory. Finally, when a block is Invalid, it means that any resident value is not valid with respect to any corresponding address indicated for the block, i.e., the value is not consistent with system memory. As seen in FIG. 2, if a block is in any of the Modified, Shared or Invalid states, it can move between the states depending upon the particular bus transaction. While a block in an Exclusive state can move to any other state, a block can only become Exclusive if it is first Invalid. A cache's block can become Invalid (e.g., from the Shared state) if the cache snoops an operation from a different processor indicating that the value held in the cache block is to be modified by the other processor, such as by snooping a read-with-intent-to-modify (RWITM) operation.

Brief Summary Text (18):
Some processor architectures, including the PowerPC.TM. processor, allow the execution of one or more special operations, other than the RWITM operation, when a processor wants to claim a memory block for a future store instruction (modifying the block). The "DClaim" operation is one example. This operation is used in lieu of the RWITM bus transaction when a valid value for the subject block is already held in the same processor's cache, e.g., in a Shared state (if the value were currently held in a Modified or Exclusive state, there would be no need to broadcast either a RWITM or DClaim request since the processor would already have exclusive control of the block). The processor may be adapted to execute a DClaim operation initially, by examining its on-board (L1) cache to see if the valid value is resident there. If not, the processor can issue a RWITM request, and any lower level cache having the valid value will, upon receiving the RWITM request, convert it into a DClaim operation to be passed to the system bus. The DClaim operation accordingly is an address-only operation since the value does not need to be read (from system memory or any intervening cache). Because of this attribute, the DClaim operation is more efficient than a RWITM operation, which would force the read operation across the system bus. When another cache has the same addressed block in a valid (Shared) state and snoops a DClaim transaction for the block, that other cache switches its corresponding block to an Invalid state, releasing the block so that the requesting processor can proceed to modify the value. In other words, a DClaim. transaction appears just like a RWITM operation from a snooper perspective.

Brief Summary Text (19):
One problem with DClaim-type operations is that they occasionally (sometimes frequently) suffer significant performance degradation, since completion of the operation can be delayed by coherency responses from other devices in the memory hierarchy. For example, if several caches of different processing units are holding a value in Shared states and they snoop a DClaim operation, their respective processors may repeatedly issue retry messages in response to the DClaim snoop (if these processors are currently busy or otherwise unable to handle the snoop, for whatever reason). This outcome means that the processor of the cache issuing the DClaim request must effectively halt processing of the associated program instruction set (an instruction "thread"), since the processor cannot complete the desired store of the modified value until the DClaim request is re-issued, possibly repeatedly, and appropriate (non-retry) responses are received from all other caches. A significant delay might also occur due to the operation having to wait in line in the cache operation queue.

Brief Summary Text (21):
If two or more caches hold a value in the Shared state and they do issue DClaim

requests simultaneously or nearly simultaneously (i.e., they attempt to cross-invalidate each other), then a deadlock can occur wherein each associated processor becomes ensnared in an endless cycle of retry responses. One method for handling such pipeline collisions is to modify the coherency protocol to preclude a cache from issuing a retry response to a DClaim request if that cache itself has an outstanding DClaim request for the same block and a response to the latter request has not been received yet. In other words, if a cache having a Shared block has issued a DClaim request and has not yet received appropriate responses allowing completion of the store instruction, and it snoops a second DClaim request from another cache for the same block, then it must not retry the second Dclaim until it has received null responses for its own request.

Brief Summary Text (29):
The foregoing objects are achieved in a method of avoiding deadlocks in a multi-processor computer system between two or more caches which are sharing a value corresponding to a system memory block, generally comprising the steps of, loading a value corresponding to a memory block of a system memory device into a plurality of cache blocks of respective processing units, assigning a first cache coherency state having a first collision priority to a single one of the plurality of cache blocks, and assigning one or more additional cache coherency states having one or more additional collision priorities which are lower than the first collision priority, to all of the remaining plurality of cache blocks other than the single cache block. A first system bus code can be issued to indicate that the first cache block is requesting modification of the memory block, while a second system bus code, different from the first system bus code, can be issued to indicate that the second cache block is requesting modification of the memory block. The invention also allows folding or elimination of certain redundant cache operations, by loading a first cache operation in the cache operations queue to request modification of a value for a cache block already held in a cache associated with the queue, loading a second cache operation in the cache operations queue to request writing of a new value for the cache block, and removing the first cache operation from the queue in response to said step of loading the second cache operation in the queue. The invention can also be applied in a global versus local manner within a multi-processor computer system having a plurality of processing units grouped into at least two clusters, each processing unit cluster having at least two cache levels wherein a given one of a plurality of caches in the first cache level is used by only a single processing unit, and a given one of a plurality of caches in the second cache level is used by two or more processing units in the same cluster, by assigning a first cache coherency state having a first collision priority to a first cache line of a cache in the first cache level associated with a first processing unit in the first processing unit cluster, and assigning one or more additional cache coherency states having one or more additional collision priorities which are lower than the first collision priority, to one or more additional cache lines of a cache in the first cache level associated with a second processing unit in the second processing unit cluster.

Detailed Description Text (7):
A cache block in the R state can effectively obtain priority in any DClaim collision by designing the coherency protocol such that any processing unit having a cache block in the S state will revoke its own pending DClaim request for that block (i.e., convert its request into a RWITM operation) if any conflicting DClaim request from an R state block is snooped before the S state block has received null responses to its own DClaim, but a processing unit will not so revoke its pending DClaim request if it has the relevant block in the R state. It may still be necessary, however, to retry a DClaim request from an R state block in certain cases, such as when the snoop queue is full and the operation cannot be snooped. However, as noted further below, the actual coherency response from the S state snooper is irrelevant with respect to the CPU that resulted in the R block DClaim operation; the main requirement is the withdrawal of any DClaim operation attempted by a cache having the block Shared, if a conflicting DClaim operation is requested

by a cache having the block Recent.

Detailed Description Text (9):
Those skilled in the art will further appreciate that the coherency response of any cache having a block in the S state is often irrelevant to a DClaim request from a corresponding block in the R state block, regardless of whether such Shared caches are issuing conflicting DClaim requests. If there is a conflicting DClaim request, it is resolved as explained above. If there is no pending DClaim request at any Shared cache, then it would issue a null response anyway, and just invalidate its block. The only time an S state block must issue a retry is if its snoop queue is full, so that it can get the operation in the snoop queue later. Accordingly, a CPU store instruction on an R state block incurs no coherency response delay, which might otherwise be particularly significant if any caches with corresponding Shared blocks were busy and had to issue repeated retry messages. Such a DClaim request may be posted (e.g., placed in a cache operations queue) for eventual broadcast to the remainder of the memory hierarchy, but the DClaim store instruction can be completed immediately, which contributes to overall faster operation of the system. This feature is particularly beneficial with multi-level cache architectures, and also results in a significant improvement in the execution of atomic read-write operations which are accomplished using load-and-reserve instructions followed by associated conditional store instructions.

Detailed Description Text (11):
In order to fully implement the foregoing, two different bus codes are provided for the two types of DClaim requests used in this embodiment, namely, the "posted" DClaim which is issued only for R state blocks, and the "regular" DClaim which is issued for S state blocks. If a given cache does not currently contain a valid copy of a value targeted by a DClaim snoop, then its snooper treats these two bus codes in exactly the same manner (for the specific cache coherency protocol depicted in FIG. 3, this situation arises only if the cache block is Invalid, so its snooper would transmit a null message in response to either type of DClaim request). Other coherency protocols may be devised in accordance with the present invention, having more than two states that are "shared" (in the broad sense that a cache block contains a current, valid copy of a value that is also present in a cache block of another processing unit). For any such coherency protocol, more than two types of DClaim requests (i.e., bus codes) may be provided in order to establish more than two different collision priority levels, further enhancing the efficiencies of the present invention.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L34: Entry 22 of 36 File: USPT Jun 6, 2000

DOCUMENT-IDENTIFIER: US 6073212 A
** See image for Certificate of Correction **
TITLE: Reducing bandwidth and areas needed for non-inclusive memory hierarchy by using dual tags

Application Filing Date (1):
19970930

Brief Summary Text (8):
For example, in an inclusive hierarchical cache unit having three levels of cache such as a level one cache, a level two cache, and a level three cache, evicting a level three cache line requires all the subblocks within the level three cache to be evicted from the level two cache and the level once cache. If every level three cache line has a size of 512 bytes, every level two cache line would have a level two cache line size of 128 bytes. Thus, there are four (4) subblocks of 128 bytes in each level three cache line which means that for each level three cache line evicted, four replacement requests are generated for the level two cache to remove any potential data copies stored in the level two cache.

Current US Cross Reference Classification (1):
711/146

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L24: Entry 7 of 11 File: USPT Aug 22, 2000

DOCUMENT-IDENTIFIER: US 6108735 A
TITLE: Method and apparatus for responding to unclaimed bus transactions

Abstract Text (1):
An agent retrieves a request, which is part of a bus transaction, from a bus. The agent then stores an identifier of the bus transaction and responds to the bus transaction after a predetermined period of time, provided the agent was not the target of the request and the target agent did not respond. In one embodiment, the agent includes a queue and a timer. A controller within the agent starts the timer if the agent is not the target of the request, a snoop phase has occurred for the request, and the request is at the top of the queue. If the timer expires without the request having received a response, then the agent responds to the request.

Application Filing Date (1):
19950929

Brief Summary Text (15):
In one embodiment, the agent includes a queue and a timer. A controller within the agent starts the timer if the agent is not the target of the request, a snoop phase has occurred for the request, and the request is at the top of the queue. If the timer expires without the request having received a response, then the agent responds to the request.

Detailed Description Text (21):
Snoop Phase

Detailed Description Text (27):
(provided there is an arbitration phase), and is two clocks long. In the first clock, an address signal is driven along with the transaction address and sufficient information to begin snooping a memory access. In the second clock, the byte enables, a transaction identifier, and the requested data transfer length are driven, along with other transaction information.

Detailed Description Text (29):
In one embodiment, every transaction that is not canceled because of an error in the error phase has a snoop phase. The snoop phase indicates if the cache line accessed in a transaction is not valid, valid or modified (dirty) in any agent's cache. In one implementation, the snoop phase is four or more clocks from the request phase.

Detailed Description Text (30):
The snoop phase of the bus defines a snoop window during which snoop events can occur on the bus. A snoop event refers to agents transmitting and/or receiving snoop results via the bus. An agent which has snoop results which need to be driven during the snoop phase drives these snoop results as a snoop event during the snoop window. All snooping agents coupled to the bus, including the agent driving the results, receive these snoop results as a snoop event during the snoop window. In one implementation, the snoop window is a single bus clock.

Detailed Description Text (31):
In one implementation, snoop results are indicated during the snoop phase using hit and modified hit signals. Assertion of the hit signal with the modified hit signal inactive indicates that a request issued on the bus hits the cache of another agent coupled to the bus. Assertion of the modified hit signal with the hit signal inactive indicates that the request issued on the bus hits the cache of another agent coupled to the bus and that the data in that cache is in a modified state. Additionally, according to the protocol used in one embodiment of the present invention, concurrent assertion of both hit and modified hit signals by an agent coupled to the bus indicates to stall the snoop pipeline of the bus. A stall on the snoop pipeline means that the present snoop phase (and the snoop pipeline) is stalled for a number of clocks, after which time it resumes its previous pace (assuming another stall is not asserted). In one implementation, a stall is for two clocks.

Detailed Description Text (36):
It is to be appreciated that, due to the pipelined nature of the bus, multiple transactions can be at different stages of the bus at different times. For example, one transaction can be in the snoop phase, while a second transaction is in the error phase, and yet a third transaction can be in the request phase. Thus, error signals and snoop signals can both be issued concurrently on the bus even though they correspond to different transactions.

Detailed Description Text (43):
In one embodiment of the present invention, the external control logic 420 also includes error correcting and detecting logic as well as external snoop logic. The error correcting and detecting logic generates Error Correcting Code (ECC) check bits for outgoing data and parity for outgoing addresses and requests, and also checks parity on incoming address, request and response pins and ECC on incoming data pins. The external snoop logic controls all snoop requests which are received from the system bus and, for processors, snoop requests which are received from the instruction fetch unit, data cache unit, or are self-generated.

Detailed Description Text (47):
The external control logic 420 is also coupled to the bus state tracking queue 430. The bus state tracking queue 430 maintains a record of the status of all transactions which are currently accessing the bus (that is, currently in the bus pipeline). The bus state tracking queue 430 includes an in-order queue 445, a snoop queue 450, and an internal control queue 455.

Detailed Description Text (48):
In one embodiment of the present invention, the bus state tracking queue 430 is a single physical queue structure which is separated logically into the in-order queue 445, the snoop queue 450, and the internal control queue 455. In this embodiment, the in-order queue 445, the snoop queue 450 and the internal control queue 455 are not physically separate queues.

Detailed Description Text (50):
The in-order queue 445 maintains a transaction target address and a transaction identifier, along with other transaction information, for each transaction in the bus state tracking queue 430. The snoop queue 450 maintains a record of whether the snoop phase for each transaction in the bus state tracking queue 430 has occurred. The internal control queue 455 tracks the state of each request in the bus state tracking queue 430 as the requests are being processed by the internal control logic 440.

Detailed Description Text (51):
The in-order queue 445, the snoop queue 450, and the internal control queue 455 are each first-in-first-out (FIFO) queues. Each time a transaction is removed from the top of the in-order queue 445, the corresponding snoop information is removed from

the top of the snoop queue 450 and the corresponding control information is removed from the top of the internal control queue 455. Thus, the correspondence of elements in the queues 445, 450 and 455 is maintained.

Detailed Description Text (53):
By way of example, the bits of the bus state tracking queue can be used in responding to unclaimed bus transactions as follows. When a request is placed on the bus, the TOR and Swc bits of the bus state tracking queue 430 are set to 0. Additionally, if the request does not target the compatibility bridge, then the TRA bit is also set to 0. The Swc bit is set to 1 by the external control logic 420 when the snoop results are known. When the request propagates to the top of the bus state tracking queue 430, then the external control logic 420 checks whether the Swc bit is set to 1 and the TRA bit is set to 0. If not, then the external control logic 420 does not start the timer 425 until these conditions are met.

Detailed Description Text (59):
The bus control logic 410 can be used to respond to unclaimed transactions, as discussed above, as well as unclaimed deferred transactions. By way of example, a request may be placed on the bus 435 and the targeted agent may issue a deferred response in the response phase. Upon receiving the deferred response, the external control logic 420 removes the request (including the original transaction identifier) from the bus state tracking queue 430 and places the request in the deferred transaction pending queue 460. When the request propagates to the top of the queue 460, the agent sets the timer 427. If a deferred response for the transaction is placed on the bus 435 (e.g., by the targeted agent) prior to expiration of the timer 427, then external control logic 420 removes the transaction from the queue 460 and resets the timer 427 (assuming there is another transaction in the queue 460). However, if the timer 427 expires, then the agent presumes that the originally targeted agent is not going to issue the deferred response and therefore the agent assumes responsibility for issuing the deferred response. The external control logic 420 generates a deferred reply transaction and arbitrates for ownership of the bus 435. Once the agent obtains ownership of the bus, the agent issues the deferred reply transaction on the bus 435 having a transaction identifier which matches the transaction identifier of the original request. The agent includes a response of either normal with data, normal without data, or hard fail, as discussed above.

Detailed Description Text (63):
However, if the bridge is not the target of the transaction then the bridge waits for the transaction to propagate to the top of the bus state tracking queue, where it checks whether the snoop phase has passed for the transaction, step 525. In one embodiment of the present invention, snoop control logic is coupled to the bus state tracking queue. In one implementation, the snoop control logic sets the Swc bit for the transaction in the bus state tracking queue when the snoop results have been sampled for the transaction. Thus, in this implementation, the control logic can determine whether the snoop phase has passed for the transaction by checking the Swc bit.

Detailed Description Text (64):
If the snoop phase for the transaction has not passed or the transaction is not at the top of the bus state tracking queue, then the present invention does not proceed with treating the transaction as an unclaimed transaction until both of these conditions are satisfied. The present invention continuously checks whether the snoop phase has passed and whether the transaction is at the top of the queue, and proceeds once both of these conditions are satisfied.

Detailed Description Text (65):
When the snoop phase has passed, the bridge starts a timer, step 530. In one embodiment, this timer is the timer 425 of FIG. 4. The bridge then checks whether the response phase for the transaction has occurred, step 535. That is, the present

invention checks whether a response has been given for this request. If the response phase has passed (and the data transfer phase, if necessary), then the bridge clears the timer and removes the request from the bus state tracking queue, step 540.

Detailed Description Text (68):
In alternate embodiments of the present invention, the bridge does not wait for the transaction to propagate to the top of the bus state tracking queue or for the snoop phase to pass before starting the timer. In these alternate embodiments, the timer can be set at different times, such as during the request phase, during the snoop phase, when the transaction propagates to the top of the bus state tracking queue (regardless of whether the snoop phase has passed), or at any other time while the transaction is pending in the bus pipeline. In these alternate embodiments, however, care should be taken to allow the targeted agent enough time to respond to the request before the present invention responds to the request as an unclaimed transaction.

Detailed Description Paragraph Table (1):
TABLE I Number Field of Bits Description Transaction 1 This bit is set to indicate Responding that the request assigned to this Agent (TRA) element is targeted to this agent. Timed Out 1 This bit is supported by the compatibility Request (TOR) bridge only. The TOR bit is activated if a transaction goes unclaimed on the bus for more than a predefined period of time. When this bit is activated, this resource is, by default, the responding agent. Snoop Window 1 This bit is set when the snoop result Complete (Swc) phase for the request assigned to this element has completed. Snoop Window 1 This bit is set when this agent needs Extension (Swe) to extend (stall) the snoop result phase. Internal 1 This bit is set by the target block Operation within the agent to indicate that Commit (loc) the data operation is complete. The loc bit may be set at the access initiation or during a later event. RequestOR 1 This bit is set if this agent initiated the (Ro) request assigned to this element. Data Buffer 1, 2 These bits contain the pointer of Pointer (Db) or 3 the data buffer element (outbound or inbound) assigned to this request. The number of bits depends on the number of data buffer elements supported. One bit is used for 1 or 2 data buffer elements, 2 bits are used for 3 or 4 elements, and 3 bits are used for 5 to 8 elements. Transfer 1 This bit specifies the direction of the Direction data transfer requested for this (Trndir) access. A 1 indicates a write and a 0 indicates a read. System 1 This bit is used to signal a Management synchronous system management Interrupt (Smi) mode (SMM) interrupt during the response phase of a transaction. For those implementations which do not support SMM, this bit can be deleted.

Detailed Description Paragraph Table (3):
TABLE II Response Type Rt[2:0] Description Configuration Address [0 0 0] This response type is not used for Cycle responding to unclaimed transactions. Retry Response [0 0 1] This response type is not used for responding to unclaimed transactions. Defer Response [0 1 0] This response type is not used for responding to unclaimed transactions. Reserved [0 1 1] Reserved Hard Failure Response [1 0 0] This response is used to respond to all unclaimed transactions when the system is configured to provide a hard failure response. Normal Response [1 0 1] This response is sent when without data responding to an unclaimed write transaction. Implicit Write Back [1 1 0] This response is sent when Response responding to an unclaimed transaction if a modified hit signal was asserted during the snoop phase for the transaction. Normal Response with [1 1 1] This response is sent when responding data to an unclaimed read transaction.

CLAIMS:

2. The system of claim 1, wherein the status of the first request includes

the first request targeting the third agent; and

a snoop phase having occurred for the first request.

8. The apparatus of claim 7, wherein the status of the first request includes the first request targeting the third agent; and

a snoop phase having occurred for the first request.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L34: Entry 35 of 36 File: USPT Apr 9, 1996

DOCUMENT-IDENTIFIER: US 5506971 A
TITLE: Method and apparatus for performing a snoop-retry protocol in a data processing system

Abstract Text (1):
A data processing system (10) and method for performing a snoop-retry protocol using an arbiter (14). Multiple bus masters (12, 16, 17) are coupled to multiple shared buses (20, 22, 24, 26). Each bus master (12, 16, 17) may initiate a bus transaction ("initiating master"), or snoop the bus transaction ("snooping bus master") occurring on a shared bus (20). When an initiating processor requests access to a dirty cache line in a memory (18), a snooping bus master asserts a shared address retry (ARTRY*) signal to inform the initiating processor to relinquish ownership of the shared bus (20) and retry the bus transaction. Upon detecting the shared ARTRY* signal, all potential bus masters remove their bus requests and ignore any bus grants from the arbiter (14), thus allowing the snooping processor which asserted the ARTRY* signal to gain ownership of the shared bus (20) to perform the snoop copyback. The arbiter (14) provides simple arbitration support to guarantee the update of the memory (18) has the highest priority among masters (12, 16, 17).

Application Filing Date (1):
19950209

Brief Summary Text (9):
In one form, the present invention comprises a data processing system and method for performing a snoop-retry bus arbitration protocol using an arbiter. The data processing system has a predetermined number of bus masters each of which is coupled to a memory system, via a predetermined number of shared buses. The arbiter is coupled to each of the predetermined number of bus masters to control allocation of bus ownership. A snooping processor asserts a first control signal in response to detecting a snoop hit on a first bus transaction. The first control signal notifies a first bus master that a data entry for an address requested during the first bus transaction is resident in the snooping processor, and to relinquish ownership of a first shared bus and retry the bus transaction. The first bus master and each of a predetermined number of potential bus masters detect assertion of the first control signal by the snooping processor, and each removes a bus request signal from the arbiter, and ignores assertion by the arbiter of a bus grant signal, in response thereto. Each of the predetermined number of potential bus masters suppresses assertion of their bus request signal to allow the snooping processor to gain ownership of the shared bus.

Detailed Description Text (9):
In accordance with the present invention, when ARTRY* is asserted, all other potential bus masters (e.g. alternate bus master 17) will remove (negate) their bus request signal and ignore any bus grant signal from arbiter 14. Each potential bus master will then continue to suppress the assertion of its BR* signal to insure that arbiter 14 receives no bus request, other than the bus request from the snooping processor (data processor 12), until the ARTRY* signal is negated. This suppression scheme allows the snooping processor (data processor 12) to acquire

mastership of the address bus 22 to write-back the modified cache line to memory 18.

Detailed Description Text (11):
On the rising edge of the fourth clock period (CLK4), data processor 12 (initiating processor) detects the assertion of the ARTRY* signal, and responds by removing (negating) its bus request (BR.sub.1 *) signal, and negating the ABB* signal to thereby release the address bus 22. Since alternate bus master 16 asserted its SSTAT* signal (in response to detecting the snoop hit), it asserts its own bus request (BR.sub.2) signal, in response to the negation of the BR.sub.1 * signal by data processor 12. As previously indicated, when ARTRY* is asserted, all other potential bus masters (i.e. alternate bus master 17) will remove (negate) their bus request signals and ignore any bus grant signal from arbiter 14. Accordingly, during the fourth clock period (CLK4), alternate bus master 17 negates its bus request (BR.sub.3 *) signal, thereby enabling the arbiter 14 to grant bus mastership to the snooping processor (alternate bus master 16).

Detailed Description Text (14):
Thus, in accordance with the present invention, upon detecting a snoop hit, a snooping processor asserts a snoop status signal (SSTAT*), which causes the assertion of a shared address retry (ARTRY*) signal. Upon receiving the ARTRY* signal, every potential bus master (except for the snooping processor attempting a memory update) removes (negates) their bus request signals, and ignores any bus grant signals already received from arbiter 14. Once the arbiter 14 detects the negation of the bus request signal by all the other potential bus masters, the arbiter 14 responds by negating any previously asserted bus grant signal. Thus, the snooping processor being the only device with an asserted bus request signal is assured ownership of the address bus 22 during the next bus tenure. The arbiter 14 asserts the bus grant signal for the snooping processor, thereby enabling the snooping processor to expeditiously perform the snoop copyback transaction.

Current US Cross Reference Classification (1):
711/146

CLAIMS:

1. In a data processing system having a predetermined number of potential bus master devices, wherein each of said predetermined number of potential bus master devices accesses an external memory system via a plurality of shared buses, a method for implementing a bus arbitration protocol using an arbiter, coupled to each of said predetermined number of potential bus master devices, said arbiter receiving a plurality of bus request signals individually generated by said predetermined number of potential bus masters, and selectively asserting one of a plurality of bus grant signals in response thereto, to control allocation of bus ownership between each of said predetermined number of potential bus masters, said method comprising the steps of:

asserting, via a first slave device, a first control signal and a second control signal in response to detecting an occurrence of a snoop hit during a first bus transaction initiated by a first master device on a first shared bus, said first control signal indicating that a data entry requested from said memory system by said first master device, during said first bus transaction, has been modified by said first slave device and that said memory system requires updating from said first slave device, said second control signal notifying said first master device to re-try said first bus transaction after said memory system has been updated by said first slave device;

detecting, via said first master device and each of said predetermined number of potential bus master devices, assertion of said second control signal by said first slave device, said first master device terminating said first bus transaction by

negating a first bus request signal, to allow said first slave device to assert a second bus request signal, and said first master device relinquishing ownership of said first shared bus by negating a third control signal in response to assertion of said second control signal, each of said predetermined number of potential bus master devices removing their individual bus request signals from said arbiter and ignoring any bus grant signal already received from said arbiter; and

suppressing assertion of said bus request signals, by each of said predetermined number of potential bus master devices, until said first slave device receives a bus grant signal from said arbiter and initiates a second bus transaction to update said memory system.

3. In a data processing system having a predetermined number of bus masters, wherein each of said predetermined number of potential bus masters accesses an external memory system via a plurality of shared buses, a method for implementing a snoop-retry bus arbitration protocol using an arbiter, coupled to each of said predetermined number of bus masters, to control allocation of bus ownership between each of said predetermined number of potential bus masters, said method comprising the steps of:

asserting, via a snooping processor, a first control signal and a second control signal in response to detecting a snoop hit during a first bus transaction initiated by a first master device on a first shared bus, said first control signal indicating that a cache line requested from said memory system, during said first bus transaction, has been modified by said snooping processor and that said memory system requires updating by said snooping processor with a current version of said cache line, said second control signal notifying said first master device to relinquish ownership of said first shared bus and retry said bus transaction after said snooping processor updates said memory system;

detecting, via said first master device and each of a predetermined number of potential bus masters, assertion of said second control signal, said first master device terminating said first bus transaction by negating a first bus request signal, and relinquishing ownership of said first shared bus by negating a third control signal in response to detecting assertion of said second control signal, said snooping processor asserting a second bus request signal to acquire bus ownership from said arbiter, each of said predetermined number of potential bus masters removing an individual bus request signal from said arbiter and ignoring any bus grant signal already received from said arbiter; and

suppressing assertion, by each of said predetermined number of potential bus masters, of said bus request signal until said arbiter grants bus ownership to said snooping processor.

7. In a data processing system having a predetermined number of bus masters each of which accesses an external memory system via a plurality of shared buses, a method for implementing a snoop-retry bus arbitration protocol using an arbiter, coupled to each of said predetermined number of bus masters, to control allocation of bus ownership between each of said plurality of shared buses, said method comprising the steps of:

monitoring, via each of a number of potential bus masters, a first bus transaction performed by a first master device on a first shared bus to detect whether an address for a requested data entry is resident in a snooping one of said potential bus masters;

providing, via said snooping one of said potential bus masters, a first control signal and a second control signal in response to said snooping one of said potential bus masters detecting that said address for said requested data entry is resident in said snooping one of said potential bus masters and that said requested

data entry stored at said address has been modified;

relinquishing, via said first master device, ownership of said first shared bus in response to receiving said second control signal by negating a third control signal, said first bus master thereafter removing a bus request signal from said arbiter;

detecting assertion of said first control signal and said second control signal, via each of said potential bus masters, and removing said bus request signal generated by each of said potential bus masters from said arbiter, in response to detecting assertion of said second control signal, each of said potential bus masters thereafter ignoring any bus grant signal previously received from said arbiter; and

suppressing assertion, by said first master device, of said bus request signal until said snooping one of said potential bus masters detects said first shared bus is available and negates said first control signal and said second control signal.

9. A data processing system, including a memory, for performing a snoop-retry bus arbitration protocol to control allocation of bus ownership for each of a predetermined number of shared buses, said data processing system comprising:

first means for detecting assertion of each of a plurality of independent bus request signals and for selectively asserting in response thereto each of a plurality of independent bus grant signals;

second means coupled to said first means and each of said predetermined number of shared buses, said second means being a first bus master of a first shared bus;

third means coupled to said first means and each of said predetermined number of shared buses, said third means snooping a first bus transaction performed by said first bus master and asserting a first control signal and a second control signal, in response to detecting that a data entry requested by said second means during said first bus transaction has been modified and a copy of said modified data entry is resident in said third means, said second control signal notifying said first bus master to relinquish ownership of said first shared bus and to retry said bus transaction after said third means updates said memory system with said modified data entry, said second means negating a first bus request signal from said first means, in response to receiving said asserted second control signal from said third means; and

fourth means coupled to said first means and each of a predetermined number of shared buses, said fourth means removing a second bus request signal from said first means in response to receiving said second control signal from said third means, said fourth means thereafter ignoring any bus grant signal already received from said first means, said second means and said fourth means suppressing re-assertion of their independent bus request signals until said third means detects said first shared bus is available and negates said first control signal and said second control signal.

Previous Doc Next Doc Go to Doc#